NESC Review of Boeing White Paper:

Research Flight Control System Software and Testing Overview - May 26, 2009

September 17, 2009

Report Approval and Revision History

Approval and Document Revision History

NOTE: This document was approved at the September 17, 2009, NRB. This document was submitted to the NESC Director on September 25, 2009, for configuration control.

Approved Version:	*Original Signature on File*	9/25/09
1.0	NESC Director	Date

Version	Description of Revision	Office of Primary Responsibility	Effective Date
1.0	Initial Release	Mr. Michael Aguilar, NASA Technical Fellow for Software at GSFC and Dr. James Stewart, the NESC Chief Engineer at DFRC	09/17/09

Table of Contents

Volume I: Technical Assessment Report

<u>List of Figures</u>

Volume II: Appendices

Volume I: Technical Assessment Report

1.0 Notification and Authorization

Mr. Gerald Budd, Project Manager of the Integrated Resilient Aircraft Control Project at Dryden Flight Research Center (DFRC), requested an independent review of a Boeing white paper for a plan to host both Level A (safety critical) software and Level B (non-safety critical) software on the Research Flight Control System (RFCS).

A NASA Engineering and Safety Center (NESC) out-of-board activity was approved April 14, 2009. Mr. Michael Aguilar, NASA Technical Fellow for Software at Goddard Space Flight Center (GSFC) and Dr. James Stewart, the NESC Chief Engineer at Dryden were selected as reviewers. The Assessment Report was presented to the NRB for approval on September 17, 2009.

The key stakeholder for this review is Integrated Resilient Aircraft Control Project (Aeronautics Division).

2.0 Signature Page

Submitted by:

Dr. James F. Stewart	Date	Mr. Michael L. Aguilar	Date

Signatories declare the findings and observations complied in the report are factually based from data extracted from Program/Project documents, contractor reports, and open literature, and/or generated from independently conducted tests, analysis, and inspections.

3.0 Team List

Name	Discipline	Organization
Core Team		
Dr. James Stewart	NESC Lead	
Mr. Michael Aguilar	NESC Deputy Lead	GSFC
Loutricia Johnson	MTSO Program Analyst	LaRC
Administrative Support		
Erin Moran	Technical Writer	LaRC/ATK

4.0 Executive Summary

The NASA Engineering and Safety Center (NESC) was requested to review a Boeing white paper dated May 26, 2009 for a plan to host both Class-A (safety critical) software and Class-B (non-safety critical) software on the same Research Flight Control System (RFCS) for the Integrated Resilient Aircraft Control (IRAC) project.

Mr. Michael Aguilar, the NASA Technical Fellow for Software at Goddard Space Flight Center (GSFC) and Dr. James Stewart, the NESC Chief Engineer at Dryden Flight Research Center (DFRC), reviewed the design approach that had been developed by the project and contractor through a review of contractor documentation and presentations, Boeing's RFCS white paper, and a series of discussions with the project and contractor. The review team assessed the application software approach, the level of testing, the methods used to separate the software, and other mitigation factors.

The RFCS is a critical part of the development of a new F-18 testbed aircraft for the IRAC Project in the Aeronautics Research Mission Directorate. The new F-18 testbed will replace the F-15 testbed that has been used for over 15 years at DFRC. The goal is to ensure that the new RFCS (based on 68040 architecture) is at least as safe as the previously flight-tested predecessor PSFCC-AARD. Refer to Appendix A to review the Boeing presentation entitled *RFCS System Software and Testing Overview*.

The following finding was identified:

F-1. The architecture of the RFCS system does provide a safe system, even with a mix of Class-A and Class-B software, if the following criteria are met:

- The "Fail-Operational Flight Control" is protected from any and all corruption by any software (Shell and Application) in-flight.
- The switch-over mechanism (hardware and/or software) is protected from any and all corruption by any software (Shell and Application) in-flight.
- The flight envelope is constrained to provide sufficient response time for detection and response to any off-nominal flight control condition by the pilot.

The following recommendation to the IRAC Project was identified:

R-1. It is recommended the memory access violation detection capability of the 68040 MMU hardware be implemented. The benefit of detecting insidious access violations using the 68040 MMU hardware would provide diagnostic capabilities advantageous during software development as well as during the flight tests.

5.0 Problem Description and Background

The RFCS currently in development for the Integrated Resilient Aircraft Controls (IRAC) project represents a new combination of components inherited from previous projects. Because the new system is created from older systems which are being reused with limited modification, it must be determined what assumptions, testing and methods were used to guarantee software safety for those previous systems. It must then be determined whether these assumptions, testing and methods remain valid and adequate for the new system, or whether this new configuration requires any modification or additions to address software safety requirements.

NESC reviewed the Boeing white paper for a plan to host both Class-A (safety critical) software and Class-B (non-safety critical) software on the same RFCS. The following were the problems and questions that needed to be addressed in the review.

The Application Software (AS) cannot meet Class-A software development and testing requirements, and still meet the requirements for rapid test and development cycles. A working solution, used on previous test architectures, is presented to limit the flight envelope such that the pilot can recover from any software induced control problem.

However, if the flight envelope is unconstrained, the Class-A requirements will need to be reviewed for all flight software development.

Problem: “AS could not meet Class-A development and testing requirements, and still meet the requirements for rapid test and development cycles.”

Question: “Does the architecture of the RFCS provide a safe system even with a mix of Class-A and Class-B software?”

Question: “What are the benefits in adding the functionality of the hardware MMU available within the 68040?”

5.1 Historical Lineage of System

The RFCS as it exists in its current form in development for the IRAC program inherits its hardware and software from several parent programs. The original implementation of the system is that of the High Alpha Research Vehicle (HARV). The research processor (a 1750 based single board computer) was installed into the flight control computer in a slot adjacent to the primary processor (the 701E Computer Memory and Timing module), sharing data through a dual-port random access memory interface. The primary processor maintains total control over the system, except during the "RFCS Engaged" state, during which it allows the research processor to take over the control law computations. This basic architecture remains unchanged to the current system.

The Production Support Flight Control Computer (PSFCC) and Fleet Support Flight Control Computer (FSFCC) took the HARV system and made it generic so that it could be used for control research in a standard F/A-18. The HARV-specific hardware (Thrust Vectoring Vanes) was removed and the 701E primary flight controls OFP modified to remove the HARV actuator interfaces, allowing RFCS control of the standard F/A-18 control surfaces. The RFCS can also command the left and right engines with throttle commands. The 1750 RFCS was programmed with a replication of the F/A-18 V10.3 control law, which was used to baseline the RFCS control law to a known state, which could then be modified for project-specific use. The PSFCC/FSFCC system was used on several projects, such as the Autonomous Formation Flight (AFF) and Automated Aerial Refueling Demonstration (AARD) projects at NASA, and the Reconfigurable Retrofit project flown by the U.S. Navy. From the time of the PSFCC project and onwards, the 701E primary flight controls OFP remained stable at PSFCC version 2.2. For this reason, this product line is referred to as PSFCC-AARD.

Part of the legacy of the current system is also derived through various projects which used a 68040-based research processor, rather than the 1750 used on the HARV/PSFCC system. A 68040-based processor board was designed for the F15/STOL Flight Control Computer in the early 1990s for the FLASH program, and used subsequently for the Advanced Control Technology for Integrated Vehicles (ACTIVE) and Intelligent Flight Controls System (IFCS) projects. The architecture is largely identical to that of the F/A-18 RFCS, with the research processor being allowed to take control of the control law function when the research is engaged. See Figure 5.2-1.

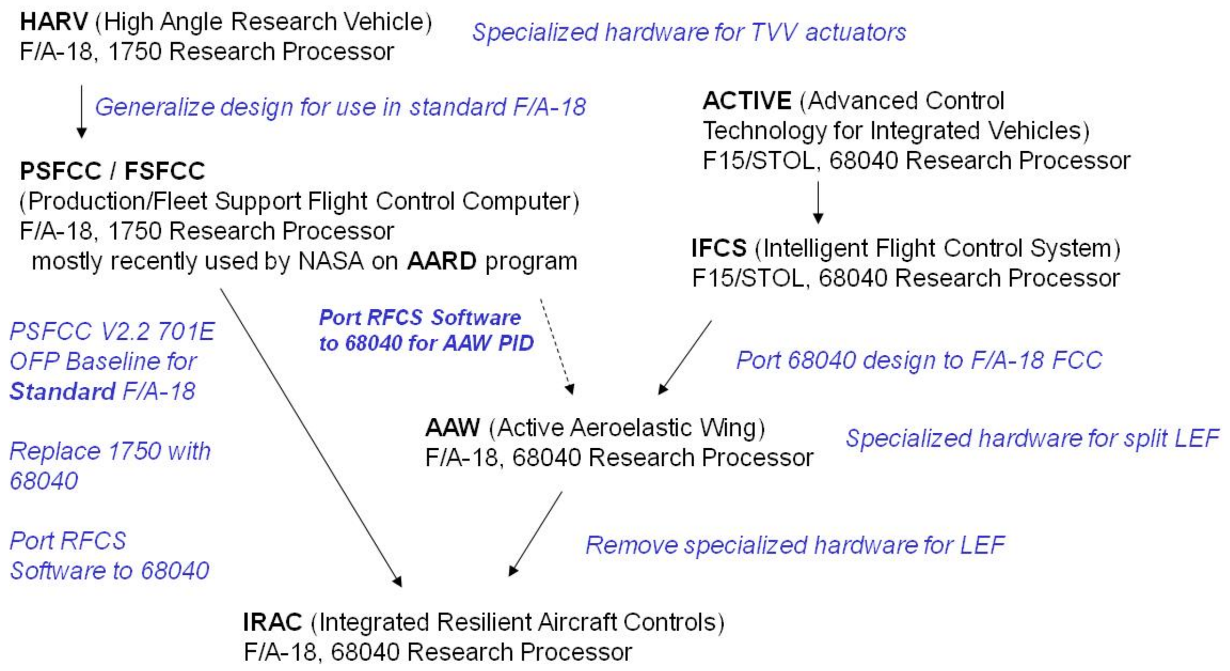

HARV (High Angle Research Vehicle)
F/A-18, 1750 Research Processor
Specialized hardware for TVV actuators
Generalize design for use in standard F/A-18
ACTIVE (Advanced Control Technology for Integrated Vehicles)
F15/STOL, 68040 Research Processor
PSFCC / FSFCC
(Production/Fleet Support Flight Control Computer)
F/A-18, 1750 Research Processor
mostly recently used by NASA on AARD program
IFCS (Intelligent Flight Control System)
F15/STOL, 68040 Research Processor
PSFCC V2.2 701E OFP Baseline for Standard F/A-18
Port RFCS Software to 68040 for AAW PID
Port 68040 design to F/A-18 FCC
AAW (Active Aeroelastic Wing)
F/A-18, 68040 Research Processor
Specialized hardware for split LEF
Replace 1750 with 68040
Port RFCS Software to 68040
Remove specialized hardware for LEF
IRAC (Integrated Resilient Aircraft Controls)
F/A-18, 68040 Research Processor

Figure 5.2-1. Historical Lineage of System

6.0 Data Analysis

The “Fail-Operational Flight Control” is defined to include all hardware and software, including control surface actuation, required for the Primary Processor to control the aircraft.

6.1 Safe-State (Fail-Operational Flight Control)

Two flight control systems are flown during a test flight. One control system exists as the “safe-state” and can be used to control the aircraft independent of the experiment. The second control system exists as the experimental application. Switch-over between the experimental application controls to the “safe-state” controls can be initiated by the pilot, by the avionics hardware, and by the experimental software. In essence, the system has fail-operational control capabilities.

Switch-Over from experimental application to Safe-State can be initiated as follows.

Pilot Initiated Switch-Over

The pilot can react to anomalous aircraft behavior by manually switching to the Fail-Operational Flight Control. The response time is dependent on the anomalous behavior and the flight envelope of the aircraft.

Hardware WDOG or COP Initiated Switch-Over

Hardware switchover can be implemented as a WDOG ("watchdog" timer timeout) or continuous heartbeat used in COP (Computer Operating Properly) implementations. Loss of a regularly scheduled signal or event initiates a switchover if the signal or event is missed.

WDOG or COP implementations detect timing and scheduling failures in both hardware and software. The detection implementation and switch-over initiation are both implemented in hardware. The response time for WDOG or COP implementations can range from milliseconds to several seconds in some systems. This response time must be sufficient to switch-over to the safe-state and maintain system safety.

In the current RFCS design, the frame cycle is used as a COP "heartbeat", and a switchover occurs in 2 milliseconds if missed.

Software Initiated Switch-Over

Software limits and consistency checks can cause a switchover initiated by the software itself. Software data validity checking can be used to detect out-of-bounds or inconsistent data values. However, data validation can miss an over-write of data that corrupts the location with a value that still passes the validity test. Software limits and consistency check access protection is based on address data content.

The detection and response time to detect and respond to limit and consistency checks is software dependent, and the switch-over is initiated in software.

Memory Access Violation Detection

Memory access violations are difficult to detect in software. Insidious memory access violations can modify data and code in a manner that can be interpreted as an algorithm failure, timing issue, math problem, etc. Detecting access violations during application software development and during flight tests would be beneficial in decerning between a coding error and a Flight Control algorithm failure.

Hardware Memory Management Unit (MMU) implementations can detect access violations. In general, access violations occur when the processor attempts fetching instructions from unintended locations, reading data from unintended locations, and writing data to unintended locations. This memory access protection is based on memory address and type of memory access.

The 68040 has the ability to protect memory segments from unintended access violations. Memory partitions are defined on a program basis by address space and access-type allowed. A task access for a particular memory partition can be defined in 68040 User's Manual Section 3.2.6 as:

- Supervisor address space protection from access by other user programs
- User address space protection from access by other user programs
- Supervisor and User address protection from any write access
- One or more pages of address space protected from write access

Memory access violations of one or more pages of memory shared between two or more programs present a problem for access violations, as this shared memory space must allow both read and write access to two or more programs.

If an access violation is detected, an exception handler, in software, is initiated and processes the fault. See 68040 User's Manual Section 5. This is a form of Software Initiated Switch-Over.

6.2 Flight Envelope

By constraining the flight envelope, the safety of the RFCS system is maintained during the initial application software testing, as listed below in steps 1 and 2. However, if and when the flight envelope is unconstrained, the pilot switch-over response time reduces the effectiveness of the manual switch-over. When the flight envelope is unconstrained, all software development must meet Class-A requirements.

1. Current RFCS
   - Class-A "Fail-Operational Flight Control"
   - Class-A Framework (shell) Software supports rapid development cycle
   - Class-B Application (non-shell) Software supports rapid development cycle
2. Initial Port to 68040
   - Class-A "Fail-Operational Flight Control"
   - Class-A Framework (shell) Software supports rapid development cycle
   - Flight Envelope constraints provide "fail operational" safety for pilot and aircraft
   - Class-B Application (non-shell) Software supports rapid development cycle
3. Future Port to 68040
   - Class-A "Fail-Operational Flight Control"
   - Class-A Framework (shell) Software supports rapid development cycle
   - Flight Envelope unconstrained
   - Class-A Application (non-shell) Software required

6.3 Conclusion

The current IRAC RFCS system provides the same level of safety as the predecessor PSFCC-AARD program, and as the Reconfigurable Retrofit program operated by NAVAIR, if the current plans for testing are followed. PSFCC-AARD was deemed safe to fly, and was successfully flight tested with no known issues relating to shell/nonshell sharing the same memory space.

The question of adding the link-only code separation for the shell and non-shell portions of the code should be addressed if the system is to proceed as is. It is not essential for the system, but would add a feature that would facilitate updates to the code in the future.
As discussed, upgrades to add robustness to the partitioning in the 68040 RFCS processor board are being considered for the future of the program.

7.0 Finding and NESC Recommendation

7.1 Finding

The "Fail-Operational Flight Control" is defined to include all hardware and software, including control surface actuation, required for the Primary Processor to control the aircraft.

The following finding was identified:

F-1. The architecture of the RFCS system does provide a safe system, even with a mix of Class-A and Class-B software, if the following criteria are met.

- The "Fail-Operational Flight Control" is protected from any and all corruption by any software (Shell and Application) in-flight.
- The switch-over mechanism (hardware and/or software) is protected from any and all corruption by any software (Shell and Application) in-flight.
- The flight envelope is constrained to provide sufficient response time for detection and response to any off-nominal flight control condition by the pilot.

7.2 NESC Recommendation

The following NESC recommendation to the IRAC Project was identified:

R-1. It is recommended the memory access violation detection capability of the 68040 MMU hardware be implemented. The benefit of detecting insidious access violations using the 68040 MMU hardware would provide diagnostic capabilities advantageous during software development as well as during the flight tests. ***(F-1)***

8.0 Other Deliverables

There are no other deliverables associated with this assessment. An Activity Closeout form will be completed after approval.

9.0 Definition of Terms

Corrective Actions	Changes to design processes, work instructions, workmanship practices, training, inspections, tests, procedures, specifications, drawings, tools, equipment, facilities, resources, or material that result in preventing, minimizing, or limiting the potential for recurrence of a problem.
Finding	A conclusion based on facts established by the investigating authority.
Lessons Learned	Knowledge or understanding gained by experience. The experience may be positive, as in a successful test or mission, or negative, as in a mishap or failure. A lesson must be significant in that it has real or assumed impact on operations; valid in that it is factually and technically correct; and applicable in that it identifies a specific design, process, or decision that reduces or limits the potential for failures and mishaps, or reinforces a positive result.
Observation	A factor, event, or circumstance identified during the assessment that did not contribute to the problem, but if left uncorrected has the potential to cause a mishap, injury, or increase the severity should a mishap occur. Alternatively, an observation could be a positive acknowledgement of a Center/Program/Project/Organization’s operational structure, tools, and/or support provided.
Problem	The subject of the independent technical assessment.
Proximate Cause	The event(s) that occurred, including any condition(s) that existed immediately before the undesired outcome, directly resulted in its occurrence and, if eliminated or modified, would have prevented the undesired outcome.
Recommendation	An action identified by the NESC to correct a root cause or deficiency identified during the investigation. The recommendations may be used by the responsible Center/Program/Project/Organization in the preparation of a corrective action plan.

Root Cause	One of multiple factors (events, conditions, or organizational factors) that contributed to or created the proximate cause and subsequent undesired outcome and, if eliminated or modified, would have prevented the undesired outcome. Typically, multiple root causes contribute to an undesired outcome.

10.0 Acronyms List

AARD	Automated Aerial Refueling Demonstration
ACTIVE	Advance Control Technology for Integrated Vehicles
AS	Application Software
COP	Computer Operating Properly
DFRC	Dryden Flight Research Center
FSFCC	Fleet Support Flight Control Computer
GSFC	Goddard Space Flight Center
HARV	High Alpha Research Vehicle
IFCS	Intelligent Flight Controls System
IRAC	Integrated Resilient Aircraft Control
MMU	Memory Management Unit
NAVAIR	Naval Air Systems Command
NESC	NASA Engineering and Safety Center
PSFCC	Production Support Flight Control Computer
RFCS	Research Flight Control System
WDOG	Watchdog (timer timeout)

11.0 References

1. AAW Final Report (AFRL-VA-WP-TR-2005-3082) Peter B. Field, Ronald K. Hess, Eric Y. Reichenbach et al, August 2005
2. Software Safety Standard (NASA STD 8719.13B), NASA Office of Safety and Mission Assurance, July 2004
3. Software Assurance Standard (NASA-STD-8739.8), NASA Office of Safety and Mission Assurance, May 2005
4. Validation Process for Mods to FSFCC OFP V316, Dr. Anthony Page, Jeffrey Monaco, Mark Hood, February 2005

5. Software Considerations in Airborne Systems and Equipment Certification (DO-178B) Radio Technical Commission for Aeronautics (RTCA), December 1992
6. Avionics Application Software Standard Interface, Part 1 – Required Services (ARINC Specification 653P1-2), Airlines Electronic Engineering Committee, March 2006
7. Final Report For Clarification OF DO-178B (DO-248B), Radio Technical Commission for Aeronautics (RTCA), October 2001
8. Autonomous Airborne Refueling Demonstrator, Software Version Description Document, RFCS Version 5.1.1, (DFRC-AARD-VDD0001.00)

Volume II: Appendices

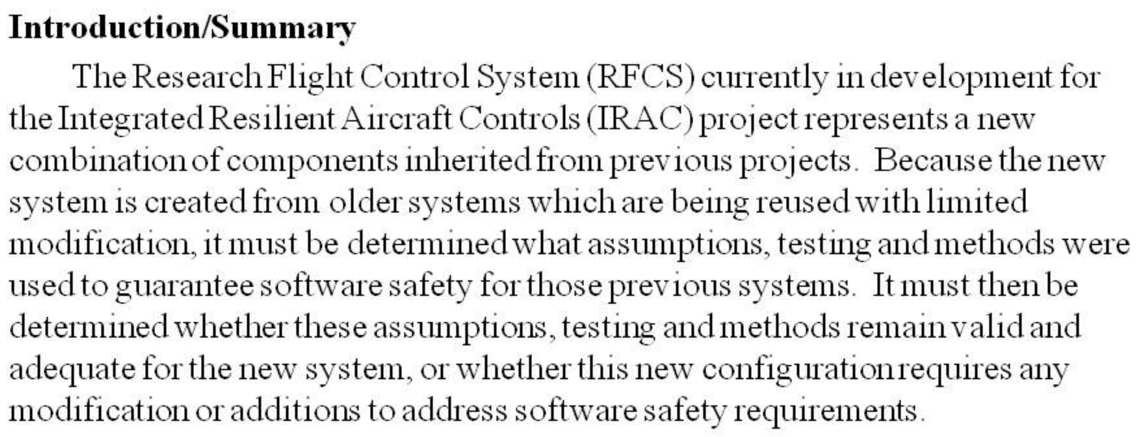
A. RFCS System Software and Testing Overview (Boeing)

Appendix A. RFCS System Software and Testing Overview (Boeing)

BOEING

Research Flight Control System
(PSFCC, AAW, IRAC)
Software and Testing Overview
Mike Stuvland

May 26, 2009 RFCS System Software and Testing Overview 1

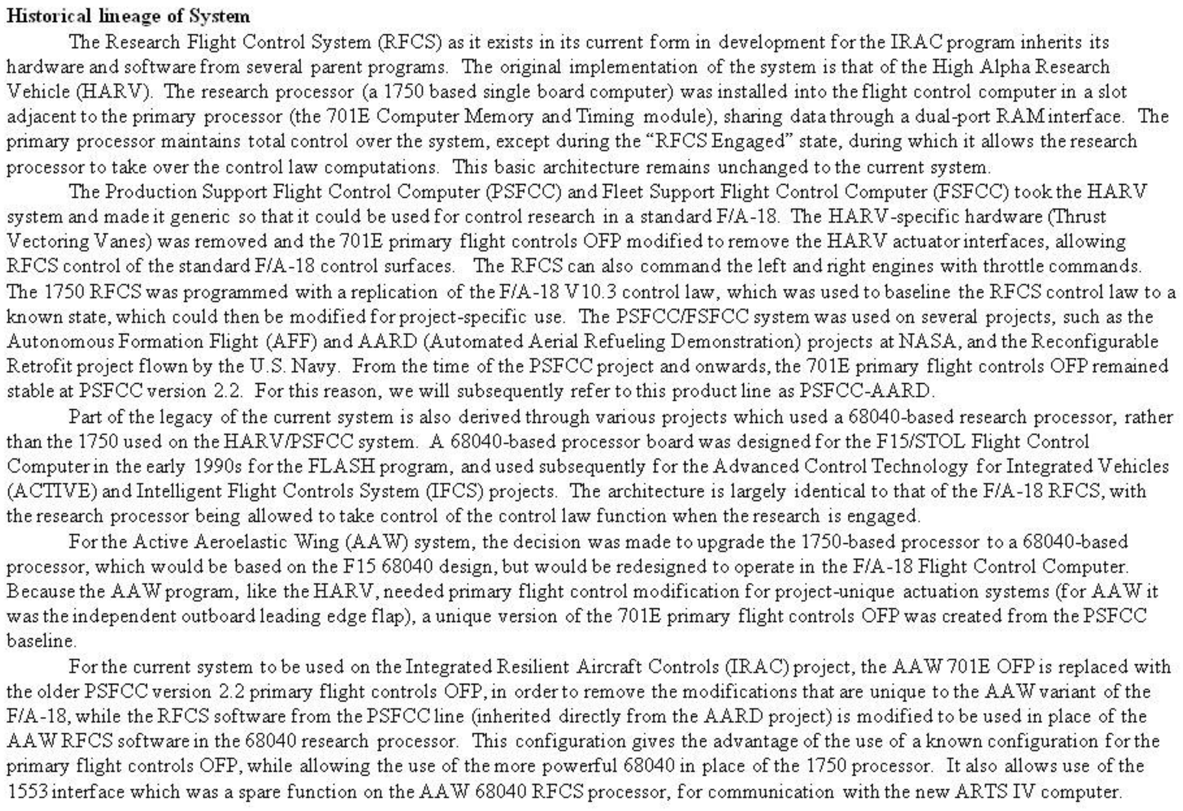


Introduction/Summary

The Research Flight Control System (RFCS) currently in development for the Integrated Resilient Aircraft Controls (IRAC) project represents a new combination of components inherited from previous projects. Because the new system is created from older systems which are being reused with limited modification, it must be determined what assumptions, testing and methods were used to guarantee software safety for those previous systems. It must then be determined whether these assumptions, testing and methods remain valid and adequate for the new system, or whether this new configuration requires any modification or additions to address software safety requirements.

Research Flight Control System
(PSFCC, AAW, IRAC)
Software and Testing Overview
Mike Stuvland

May 26, 2009 RFCS System Software and Testing Overview 1



Historical lineage of System

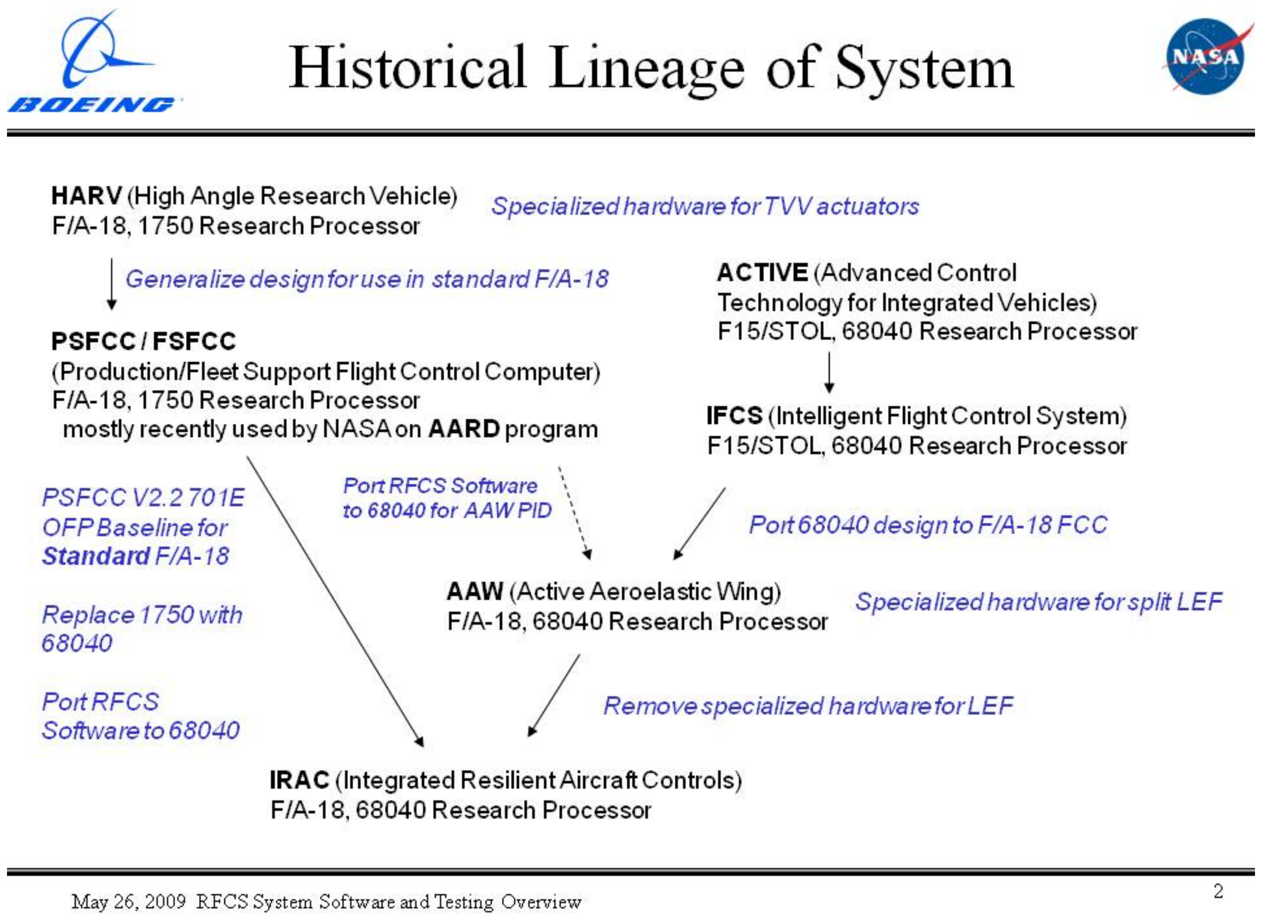
The Research Flight Control System (RFCS) as it exists in its current form in development for the IRAC program inherits its hardware and software from several parent programs. The original implementation of the system is that of the High Alpha Research Vehicle (HARV). The research processor (a 1750 based single board computer) was installed into the flight control computer in a slot adjacent to the primary processor (the 701E Computer Memory and Timing module), sharing data through a dual-port RAM interface. The primary processor maintains total control over the system, except during the "RFCS Engaged" state, during which it allows the research processor to take over the control law computations. This basic architecture remains unchanged to the current system.

The Production Support Flight Control Computer (PSFCC) and Fleet Support Flight Control Computer (FSFCC) took the HARV system and made it generic so that it could be used for control research in a standard F/A-18. The HARV-specific hardware (Thrust Vectoring Vanes) was removed and the 701E primary flight controls OFP modified to remove the HARV actuator interfaces, allowing RFCS control of the standard F/A-18 control surfaces. The RFCS can also command the left and right engines with throttle commands. The 1750 RFCS was programmed with a replication of the F/A-18 V10.3 control law, which was used to baseline the RFCS control law to a known state, which could then be modified for project-specific use. The PSFCC/FSFCC system was used on several projects, such as the Autonomous Formation Flight (AFF) and AARD (Automated Aerial Refueling Demonstration) projects at NASA, and the Reconfigurable Retrofit project flown by the U.S. Navy. From the time of the PSFCC project and onwards, the 701E primary flight controls OFP remained stable at PSFCC version 2.2. For this reason, we will subsequently refer to this product line as PSFCC-AARD.

Part of the legacy of the current system is also derived through various projects which used a 68040-based research processor, rather than the 1750 used on the HARV/PSFCC system. A 68040-based processor board was designed for the F15/STOL Flight Control Computer in the early 1990s for the FLASH program, and used subsequently for the Advanced Control Technology for Integrated Vehicles (ACTIVE) and Intelligent Flight Controls System (IFCS) projects. The architecture is largely identical to that of the F/A-18 RFCS, with the research processor being allowed to take control of the control law function when the research is engaged.

For the Active Aeroelastic Wing (AAW) system, the decision was made to upgrade the 1750-based processor to a 68040-based processor, which would be based on the F15 68040 design, but would be redesigned to operate in the F/A-18 Flight Control Computer. Because the AAW program, like the HARV, needed primary flight control modification for project-unique actuation systems (for AAW it was the independent outboard leading edge flap), a unique version of the 701E primary flight controls OFP was created from the PSFCC baseline.

For the current system to be used on the Integrated Resilient Aircraft Controls (IRAC) project, the AAW 701E OFP is replaced with the older PSFCC version 2.2 primary flight controls OFP, in order to remove the modifications that are unique to the AAW variant of the F/A-18, while the RFCS software from the PSFCC line (inherited directly from the AARD project) is modified to be used in place of the AAW RFCS software in the 68040 research processor. This configuration gives the advantage of the use of a known configuration for the primary flight controls OFP, while allowing the use of the more powerful 68040 in place of the 1750 processor. It also allows use of the 1553 interface which was a spare function on the AAW 68040 RFCS processor, for communication with the new ARTS IV computer.



Historical Lineage of System

HARV (High Angle Research Vehicle)
F/A-18, 1750 Research Processor

Specialized hardware for TVV actuators

Generalize design for use in standard F/A-18

PSFCC / FSFCC
(Production/Fleet Support Flight Control Computer)
F/A-18, 1750 Research Processor
mostly recently used by NASA on **AARD** program

ACTIVE (Advanced Control Technology for Integrated Vehicles)
F15/STOL, 68040 Research Processor

IFCS (Intelligent Flight Control System)
F15/STOL, 68040 Research Processor

*PSFCC V2.2 701E OFP Baseline for **Standard** F/A-18*

Replace 1750 with 68040

Port RFCS Software to 68040

Port RFCS Software to 68040 for AAW PID

Port 68040 design to F/A-18 FCC

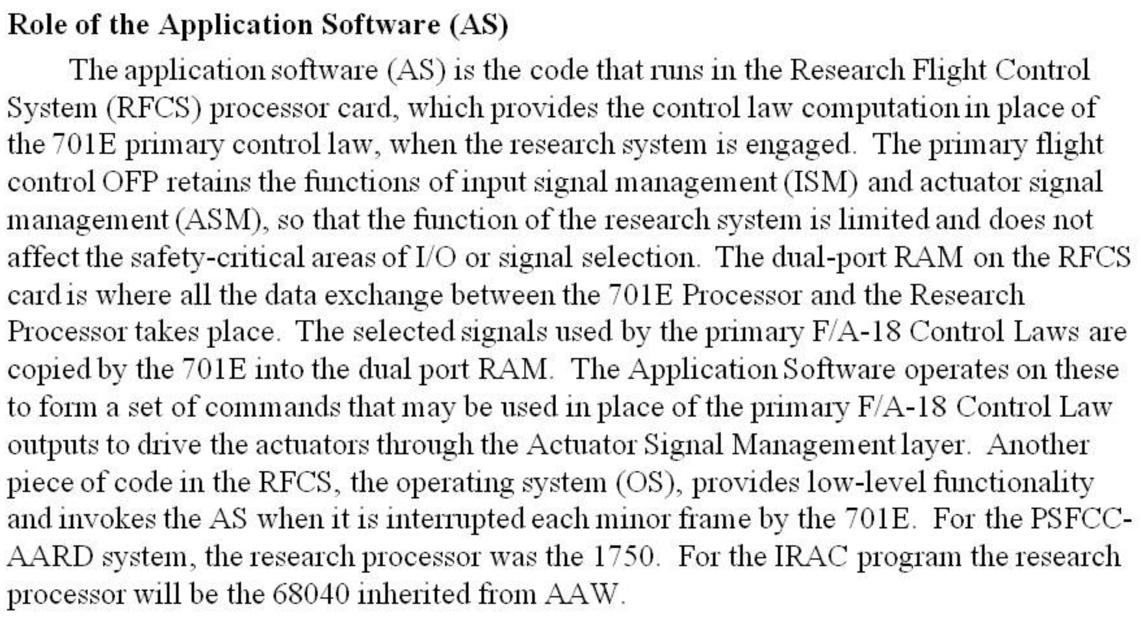
AAW (Active Aeroelastic Wing)
F/A-18, 68040 Research Processor

Specialized hardware for split LEF

Remove specialized hardware for LEF

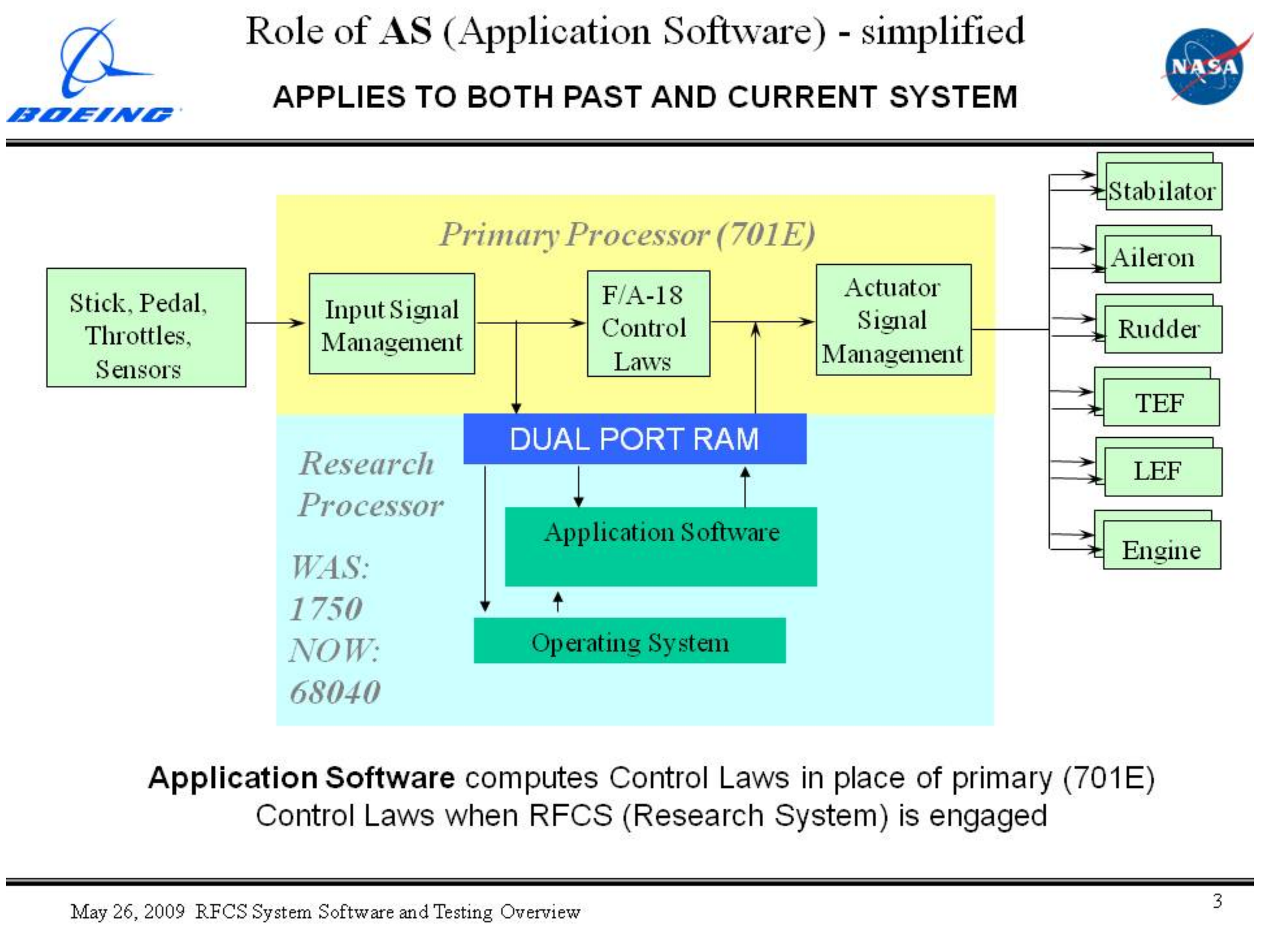
IRAC (Integrated Resilient Aircraft Controls)
F/A-18, 68040 Research Processor

May 26, 2009 RFCS System Software and Testing Overview 2



Role of the Application Software (AS)

The application software (AS) is the code that runs in the Research Flight Control System (RFCS) processor card, which provides the control law computation in place of the 701E primary control law, when the research system is engaged. The primary flight control OFP retains the functions of input signal management (ISM) and actuator signal management (ASM), so that the function of the research system is limited and does not affect the safety-critical areas of I/O or signal selection. The dual-port RAM on the RFCS card is where all the data exchange between the 701E Processor and the Research Processor takes place. The selected signals used by the primary F/A-18 Control Laws are copied by the 701E into the dual port RAM. The Application Software operates on these to form a set of commands that may be used in place of the primary F/A-18 Control Law outputs to drive the actuators through the Actuator Signal Management layer. Another piece of code in the RFCS, the operating system (OS), provides low-level functionality and invokes the AS when it is interrupted each minor frame by the 701E. For the PSFCC-AARD system, the research processor was the 1750. For the IRAC program the research processor will be the 68040 inherited from AAW.

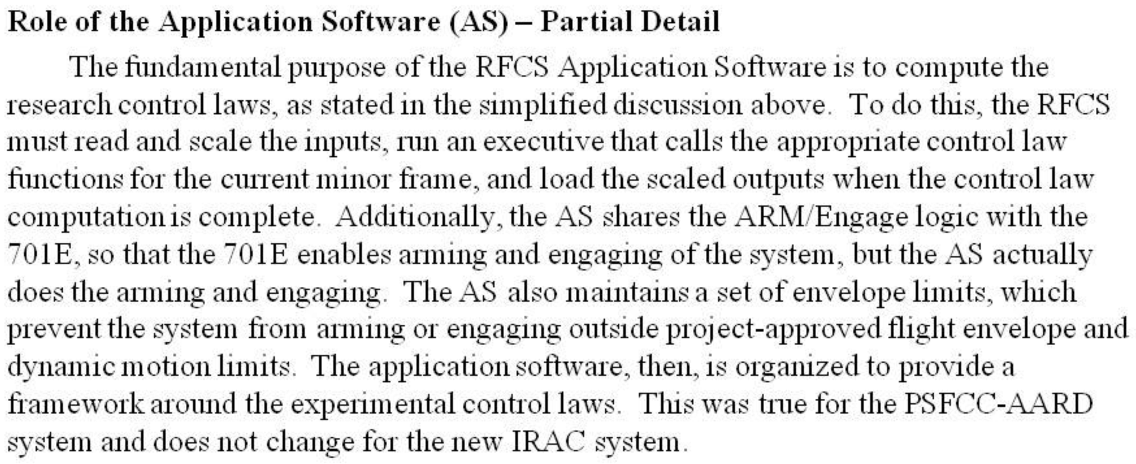


Role of AS (Application Software) - simplified

APPLIES TO BOTH PAST AND CURRENT SYSTEM

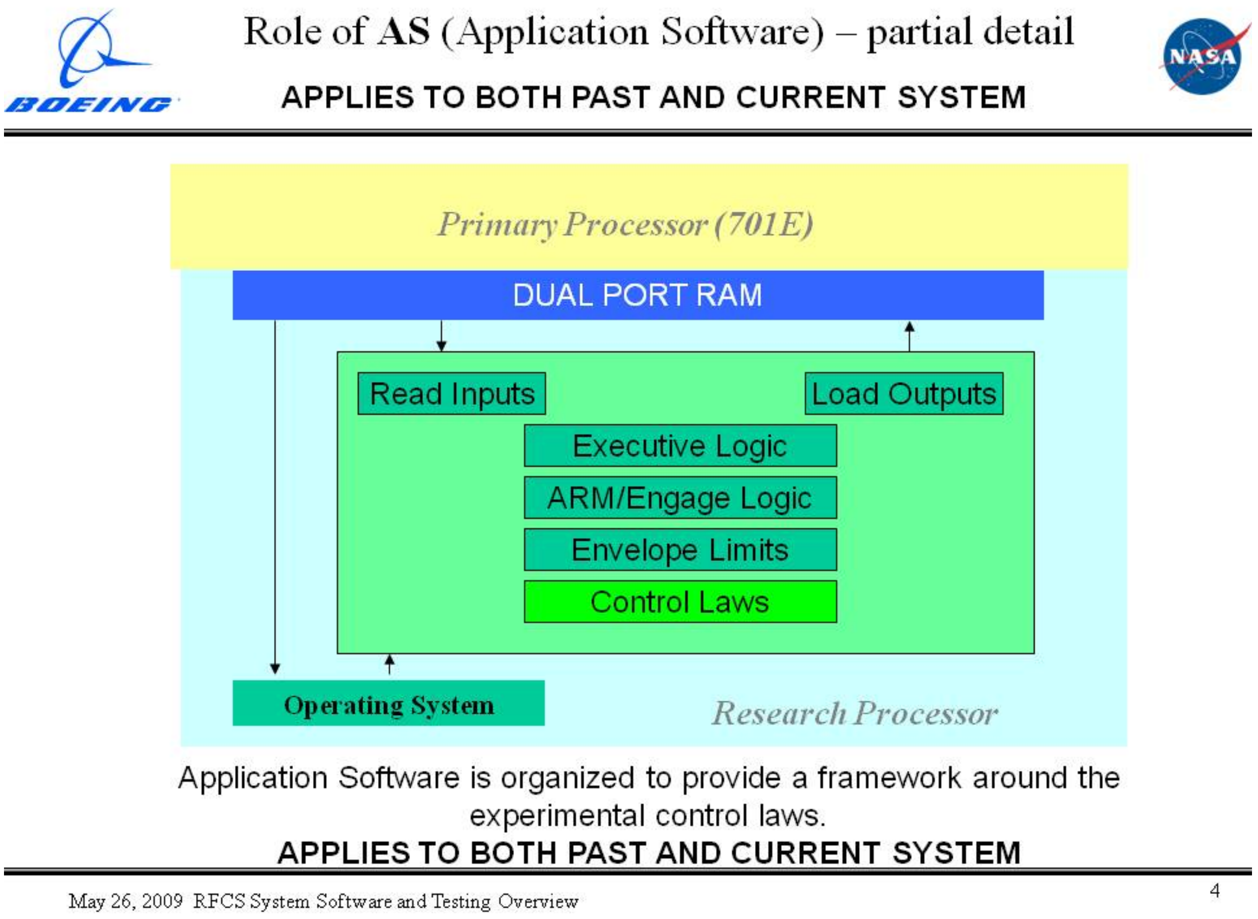
Application Software computes Control Laws in place of primary (701E) Control Laws when RFCS (Research System) is engaged

May 26, 2009 RFCS System Software and Testing Overview 3



Role of the Application Software (AS) – Partial Detail

The fundamental purpose of the RFCS Application Software is to compute the research control laws, as stated in the simplified discussion above. To do this, the RFCS must read and scale the inputs, run an executive that calls the appropriate control law functions for the current minor frame, and load the scaled outputs when the control law computation is complete. Additionally, the AS shares the ARM/Engage logic with the 701E, so that the 701E enables arming and engaging of the system, but the AS actually does the arming and engaging. The AS also maintains a set of envelope limits, which prevent the system from arming or engaging outside project-approved flight envelope and dynamic motion limits. The application software, then, is organized to provide a framework around the experimental control laws. This was true for the PSFCC-AARD system and does not change for the new IRAC system.



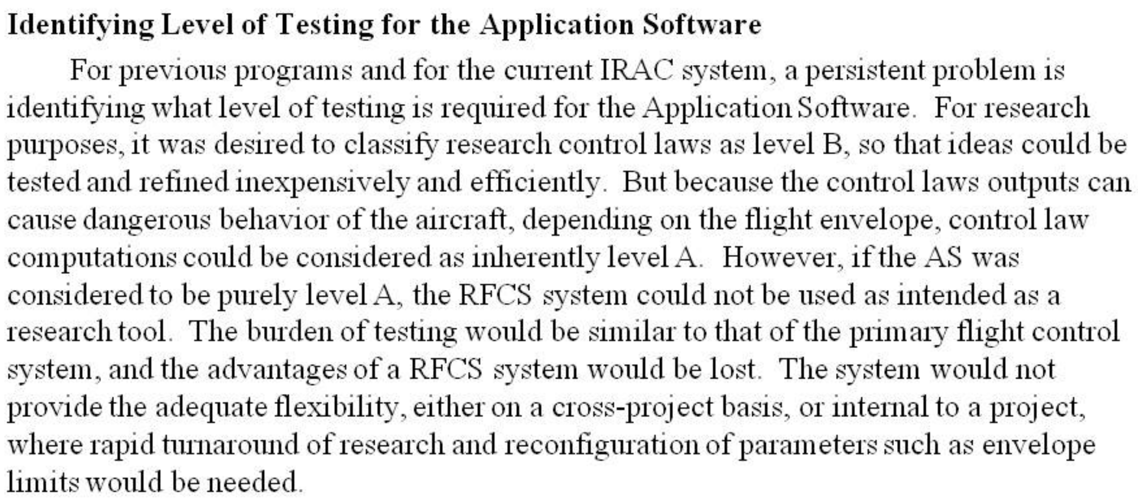
Role of AS (Application Software) – partial detail

APPLIES TO BOTH PAST AND CURRENT SYSTEM

Application Software is organized to provide a framework around the experimental control laws.

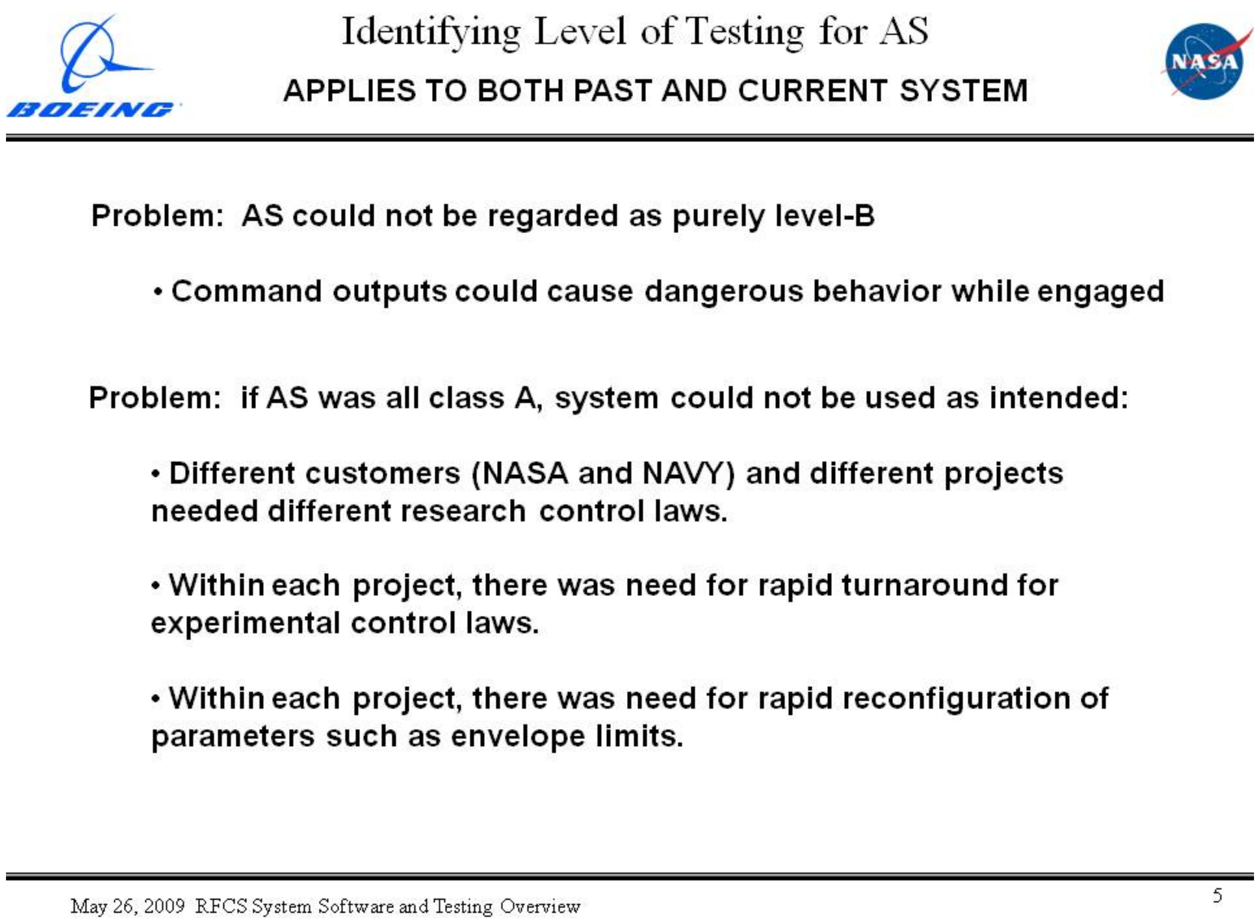
APPLIES TO BOTH PAST AND CURRENT SYSTEM

May 26, 2009 RFCS System Software and Testing Overview 4



Identifying Level of Testing for the Application Software

For previous programs and for the current IRAC system, a persistent problem is identifying what level of testing is required for the Application Software. For research purposes, it was desired to classify research control laws as level B, so that ideas could be tested and refined inexpensively and efficiently. But because the control laws outputs can cause dangerous behavior of the aircraft, depending on the flight envelope, control law computations could be considered as inherently level A. However, if the AS was considered to be purely level A, the RFCS system could not be used as intended as a research tool. The burden of testing would be similar to that of the primary flight control system, and the advantages of a RFCS system would be lost. The system would not provide the adequate flexibility, either on a cross-project basis, or internal to a project, where rapid turnaround of research and reconfiguration of parameters such as envelope limits would be needed.



Identifying Level of Testing for AS

APPLIES TO BOTH PAST AND CURRENT SYSTEM

Problem: AS could not be regarded as purely level-B

- Command outputs could cause dangerous behavior while engaged

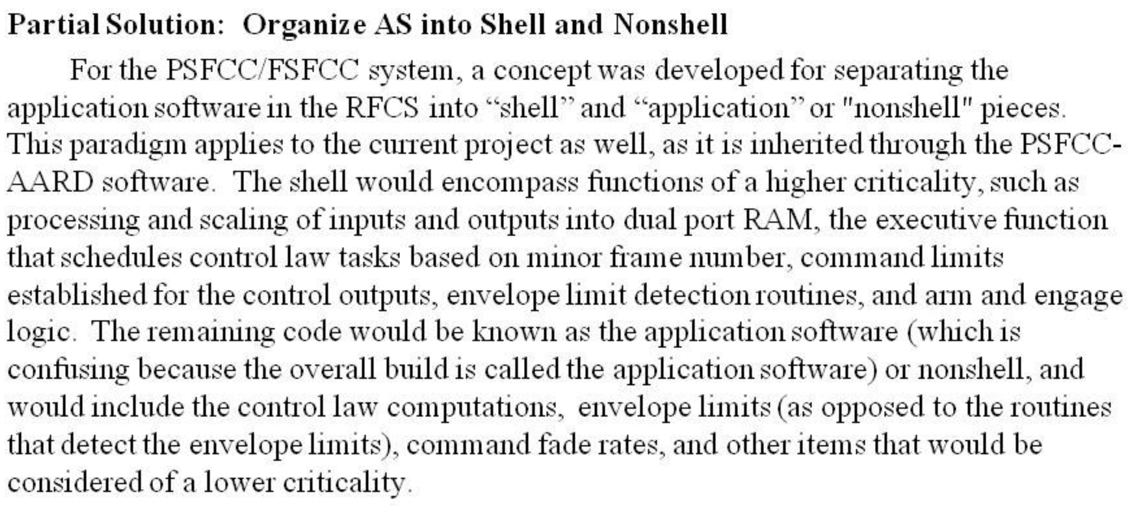
Problem: if AS was all class A, system could not be used as intended:

- Different customers (NASA and NAVY) and different projects needed different research control laws.

- Within each project, there was need for rapid turnaround for experimental control laws.

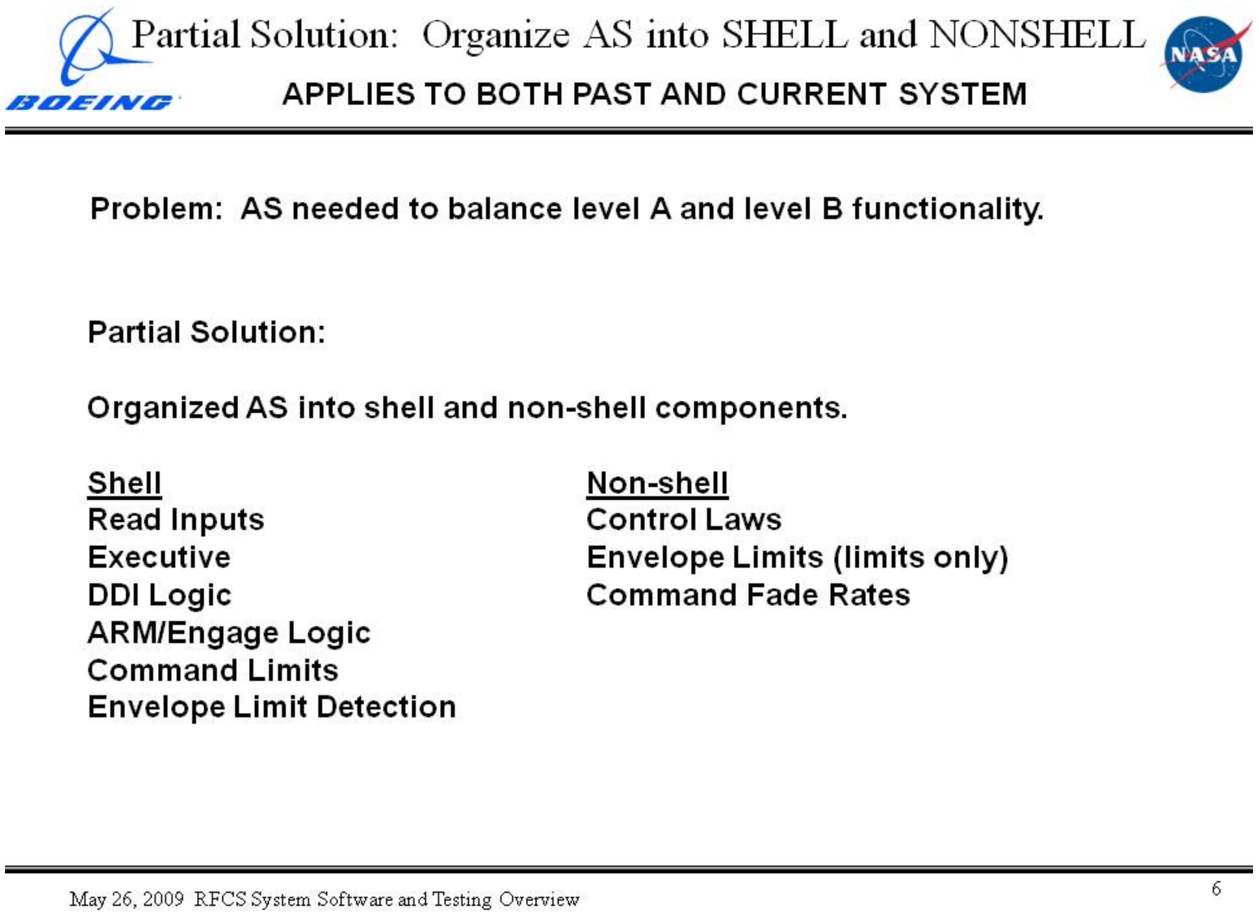
- Within each project, there was need for rapid reconfiguration of parameters such as envelope limits.

May 26, 2009 RFCS System Software and Testing Overview 5



Partial Solution: Organize AS into Shell and Nonshell

For the PSFCC/FSFCC system, a concept was developed for separating the application software in the RFCS into “shell” and “application” or "nonshell" pieces. This paradigm applies to the current project as well, as it is inherited through the PSFCC-AARD software. The shell would encompass functions of a higher criticality, such as processing and scaling of inputs and outputs into dual port RAM, the executive function that schedules control law tasks based on minor frame number, command limits established for the control outputs, envelope limit detection routines, and arm and engage logic. The remaining code would be known as the application software (which is confusing because the overall build is called the application software) or nonshell, and would include the control law computations, envelope limits (as opposed to the routines that detect the envelope limits), command fade rates, and other items that would be considered of a lower criticality.



Partial Solution: Organize AS into SHELL and NONSHELL

APPLIES TO BOTH PAST AND CURRENT SYSTEM

Problem: AS needed to balance level A and level B functionality.

Partial Solution:

Organized AS into shell and non-shell components.

Shell	Non-shell
Read Inputs	Control Laws
Executive	Envelope Limits (limits only)
DDI Logic	Command Fade Rates
ARM/Engage Logic	
Command Limits	
Envelope Limit Detection	

May 26, 2009 RFCS System Software and Testing Overview 6

The Purpose of the Shell – minimal partition

The following quote from the AAW Final Report (Boeing Company, 2004) provides good insight into what the shell was meant to be:

"Shell software was designed to provide the framework for the software and remain unchanged from software build to software build. The application software would provide the code for the desired experiment."

The separation of the shell from the nonshell code was a configuration construct, which served to identify portions of code that would remain static between builds, and provide some basis for which level of testing (level A or level B) was to be applied to that code. If it could be shown that the shell did not change between builds, and that only nonshell elements had been modified, a limited form of verification would be required for a new AS release. No claim was made of a "robust" type of safety partition that would have prevented the level B code from full access to memory or the ability to improperly execute level A code, in the case of a programming error or other fault in the level B code, as would be strictly required in a DO-178B certified partitioned system (see D0-248B, section DP 4.14, for a detailed discussion of what is entailed in robust portioning), or a system that formally met the NASA requirements for partitioning (ref. NASA-STD-8719.13B, section 4.1.1.2, note 4-2).

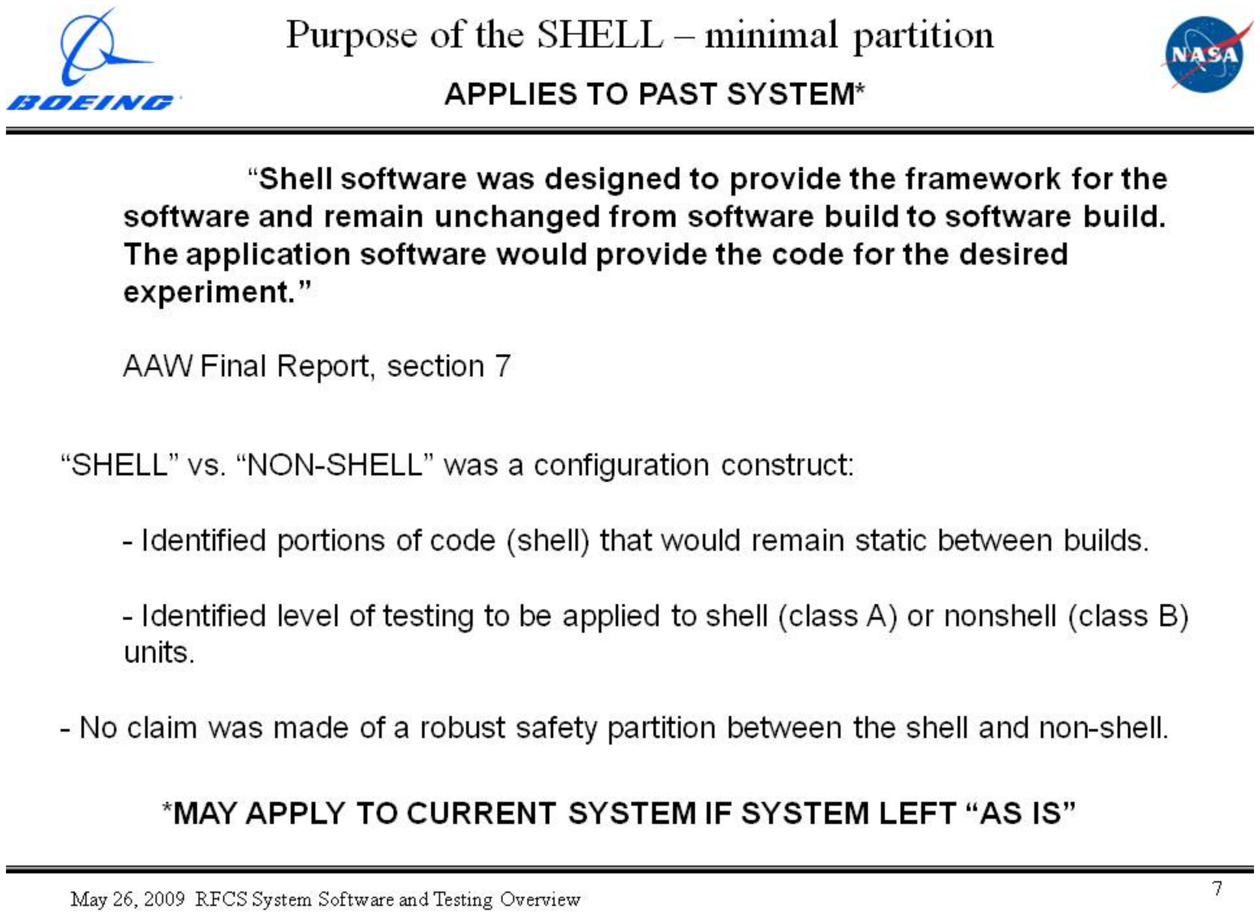
The type of minimal software-only partition that was implemented via the shell/nonshell separation is described in a section entitled "Partitioning Rigor" of DO-248B, which is a document whose purpose is to clarify topics in DO-178B:

> A less rigorous form of partitioning, for lower software levels, is to use only compiler and linker mechanisms to implement partitioning. This will lead to partitioning by design, but the user has to rely on the correctness of these tools (DO-248B, section DP-4.14.3).

In practice, the standard for robust partitioning is often ARINC-653, which outlines a system that meets the standards of space, time, and control coupling described in DO-178B/DO-248B. For adherents of systems which perform to ARINC-653 or similar standard, it may not appear that the "software-only" partitioning provided by the shell/nonshell constituted what was normally considered partitioning at all.

For PSFCC-AARD, the shell and nonshell were linked to different regions in PROM memory, so that a process of binary comparison would show that the shell had been unaffected by the change, since all shell modules had been located in the shell region which could be shown by checksum or bit-by-bit comparison not to have been changed. This did not guarantee that the nonshell portion of the software could not, theoretically, interfere with the execution of the shell software; but it did give a reasonable level of confidence that the operation of the shell would remain unaffected due to a change in the nonshell, and that the scope of the changes in the resultant build were well-known and isolated to the targeted function.

The purpose of the shell has not been altered from the PSFCC build for the new IRAC system, although changes are being considered to enhance the partitioning within the RFCS.



Purpose of the SHELL – minimal partition

APPLIES TO PAST SYSTEM*

"Shell software was designed to provide the framework for the software and remain unchanged from software build to software build. The application software would provide the code for the desired experiment."

AAW Final Report, section 7

"SHELL" vs. "NON-SHELL" was a configuration construct:

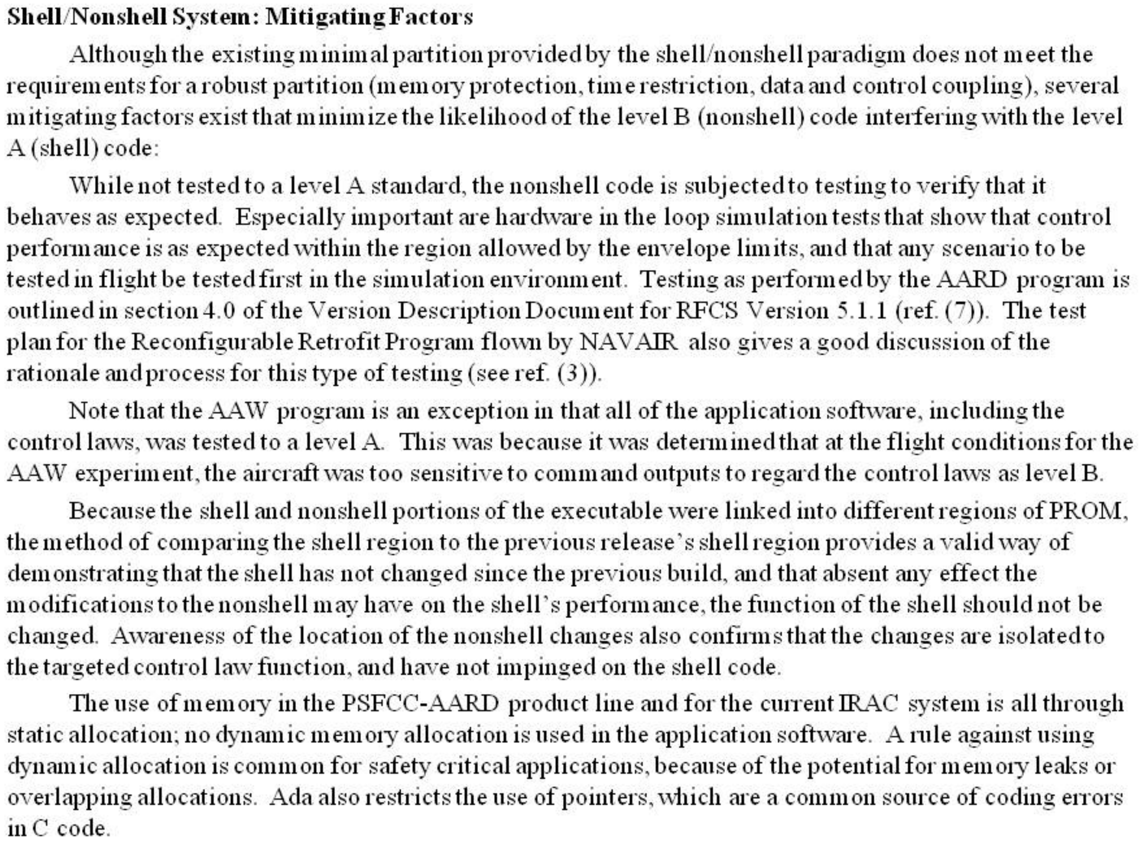
- Identified portions of code (shell) that would remain static between builds.

- Identified level of testing to be applied to shell (class A) or nonshell (class B) units.

- No claim was made of a robust safety partition between the shell and non-shell.

***MAY APPLY TO CURRENT SYSTEM IF SYSTEM LEFT "AS IS"**

May 26, 2009 RFCS System Software and Testing Overview 7



Shell/Nonshell System: Mitigating Factors

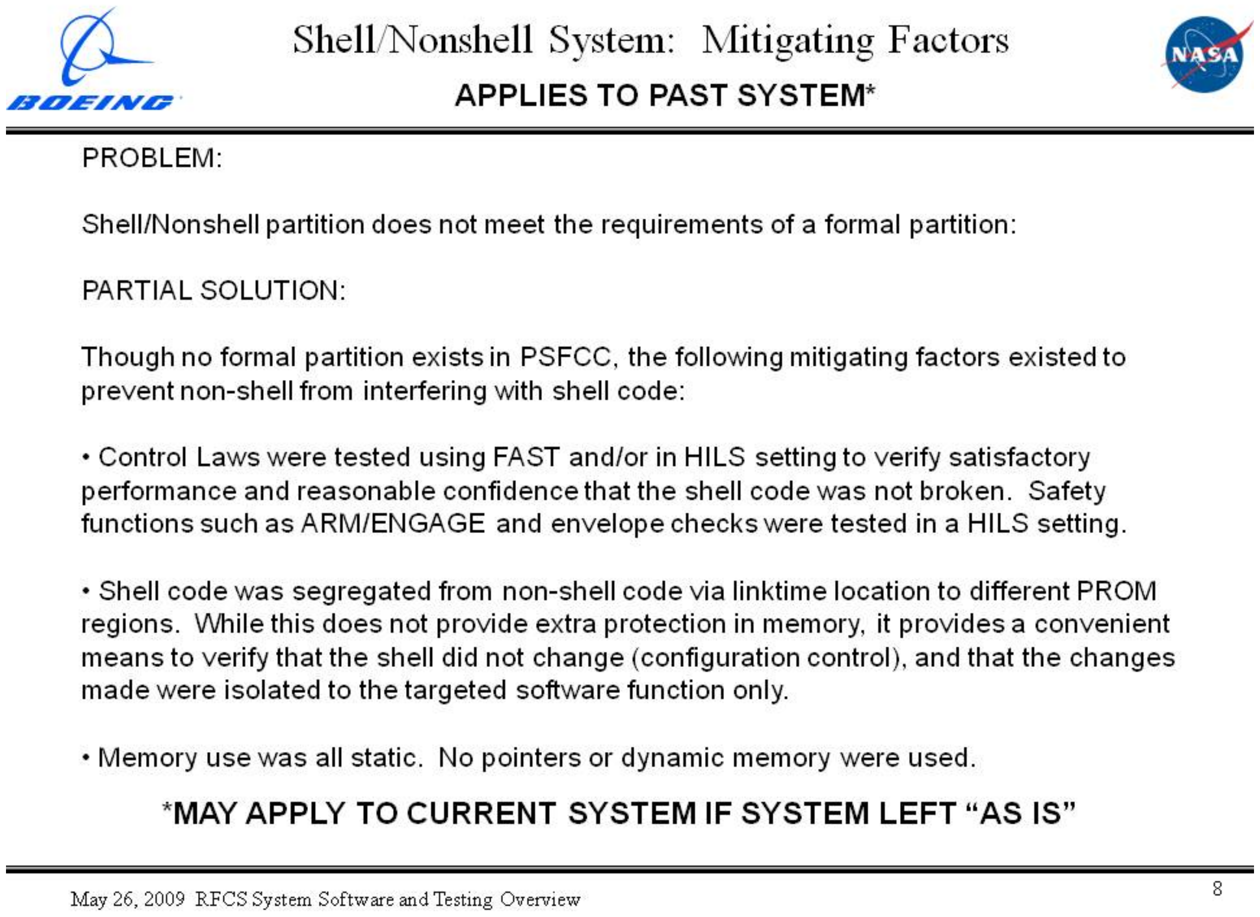
Although the existing minimal partition provided by the shell/nonshell paradigm does not meet the requirements for a robust partition (memory protection, time restriction, data and control coupling), several mitigating factors exist that minimize the likelihood of the level B (nonshell) code interfering with the level A (shell) code:

While not tested to a level A standard, the nonshell code is subjected to testing to verify that it behaves as expected. Especially important are hardware in the loop simulation tests that show that control performance is as expected within the region allowed by the envelope limits, and that any scenario to be tested in flight be tested first in the simulation environment. Testing as performed by the AARD program is outlined in section 4.0 of the Version Description Document for RFCS Version 5.1.1 (ref. (7)). The test plan for the Reconfigurable Retrofit Program flown by NAVAIR also gives a good discussion of the rationale and process for this type of testing (see ref. (3)).

Note that the AAW program is an exception in that all of the application software, including the control laws, was tested to a level A. This was because it was determined that at the flight conditions for the AAW experiment, the aircraft was too sensitive to command outputs to regard the control laws as level B.

Because the shell and nonshell portions of the executable were linked into different regions of PROM, the method of comparing the shell region to the previous release's shell region provides a valid way of demonstrating that the shell has not changed since the previous build, and that absent any effect the modifications to the nonshell may have on the shell's performance, the function of the shell should not be changed. Awareness of the location of the nonshell changes also confirms that the changes are isolated to the targeted control law function, and have not impinged on the shell code.

The use of memory in the PSFCC-AARD product line and for the current IRAC system is all through static allocation; no dynamic memory allocation is used in the application software. A rule against using dynamic allocation is common for safety critical applications, because of the potential for memory leaks or overlapping allocations. Ada also restricts the use of pointers, which are a common source of coding errors in C code.



Shell/Nonshell System: Mitigating Factors

APPLIES TO PAST SYSTEM*

PROBLEM:

Shell/Nonshell partition does not meet the requirements of a formal partition:

PARTIAL SOLUTION:

Though no formal partition exists in PSFCC, the following mitigating factors existed to prevent non-shell from interfering with shell code:

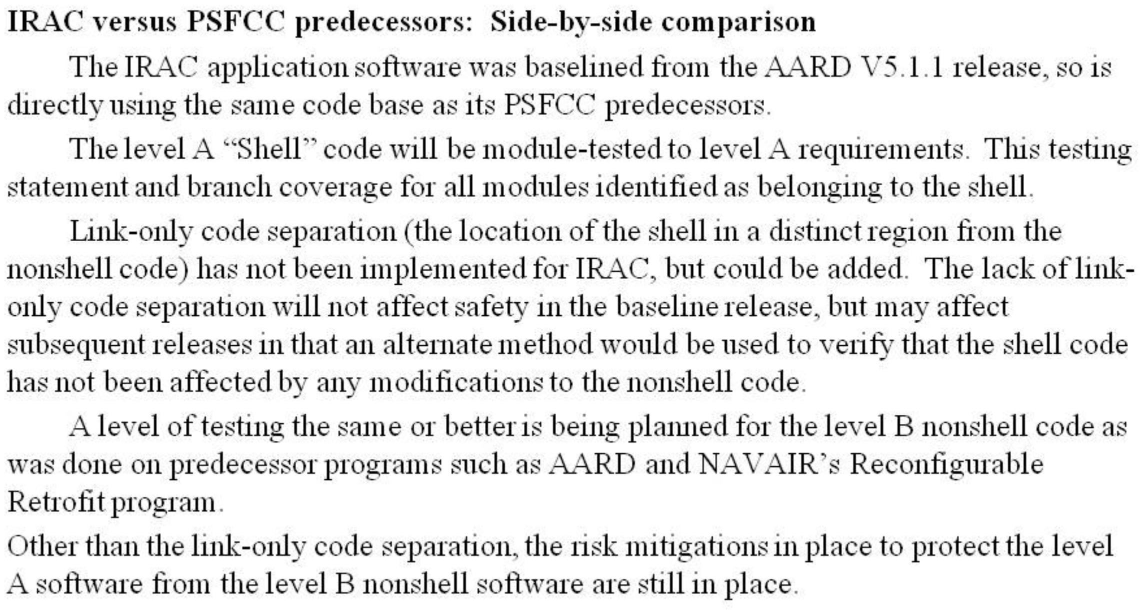
• Control Laws were tested using FAST and/or in HILS setting to verify satisfactory performance and reasonable confidence that the shell code was not broken. Safety functions such as ARM/ENGAGE and envelope checks were tested in a HILS setting.

• Shell code was segregated from non-shell code via linktime location to different PROM regions. While this does not provide extra protection in memory, it provides a convenient means to verify that the shell did not change (configuration control), and that the changes made were isolated to the targeted software function only.

• Memory use was all static. No pointers or dynamic memory were used.

***MAY APPLY TO CURRENT SYSTEM IF SYSTEM LEFT "AS IS"**

May 26, 2009 RFCS System Software and Testing Overview 8



IRAC versus PSFCC predecessors: Side-by-side comparison

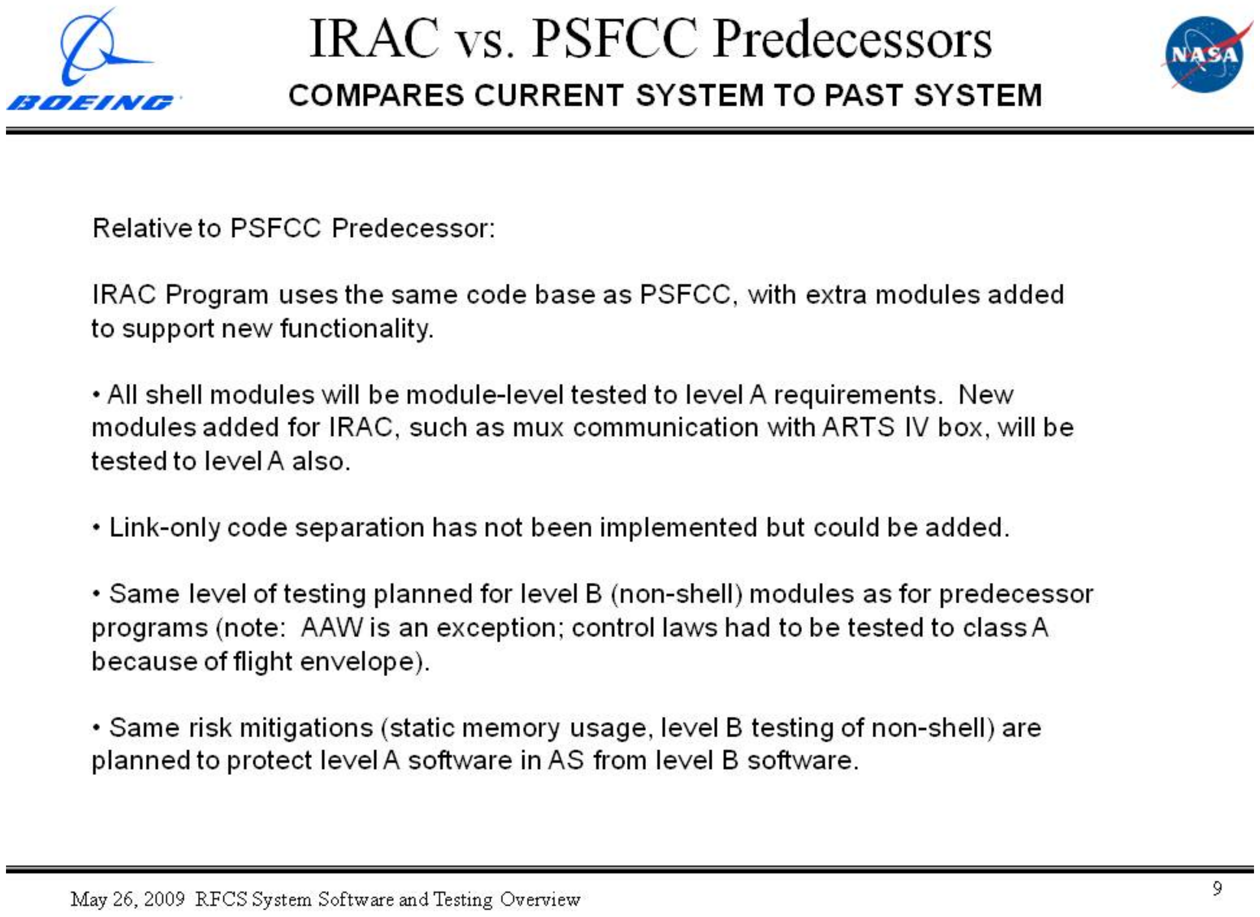
The IRAC application software was baselined from the AARD V5.1.1 release, so is directly using the same code base as its PSFCC predecessors.

The level A "Shell" code will be module-tested to level A requirements. This testing statement and branch coverage for all modules identified as belonging to the shell.

Link-only code separation (the location of the shell in a distinct region from the nonshell code) has not been implemented for IRAC, but could be added. The lack of link-only code separation will not affect safety in the baseline release, but may affect subsequent releases in that an alternate method would be used to verify that the shell code has not been affected by any modifications to the nonshell code.

A level of testing the same or better is being planned for the level B nonshell code as was done on predecessor programs such as AARD and NAVAIR's Reconfigurable Retrofit program.

Other than the link-only code separation, the risk mitigations in place to protect the level A software from the level B nonshell software are still in place.



IRAC vs. PSFCC Predecessors

COMPARES CURRENT SYSTEM TO PAST SYSTEM

Relative to PSFCC Predecessor:

IRAC Program uses the same code base as PSFCC, with extra modules added to support new functionality.

• All shell modules will be module-level tested to level A requirements. New modules added for IRAC, such as mux communication with ARTS IV box, will be tested to level A also.

• Link-only code separation has not been implemented but could be added.

• Same level of testing planned for level B (non-shell) modules as for predecessor programs (note: AAW is an exception; control laws had to be tested to class A because of flight envelope).

• Same risk mitigations (static memory usage, level B testing of non-shell) are planned to protect level A software in AS from level B software.

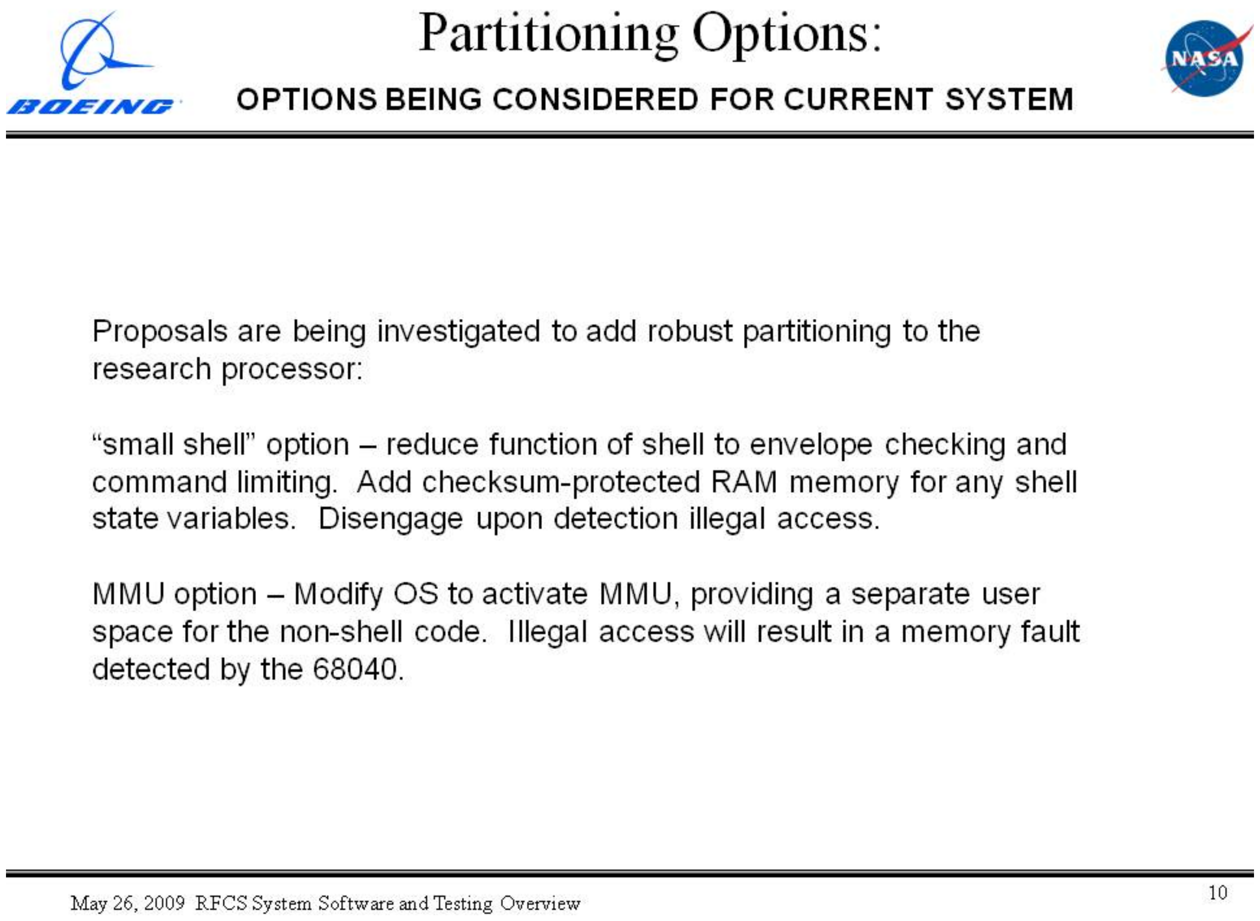
May 26, 2009 RFCS System Software and Testing Overview 9

Partitioning Options:

For the current IRAC project, proposals have been forwarded to add formal partitioning to the 68040 RFCS processor board that would meet some if not all of the requirements of a partitioned system as defined for DO-178B, and clarified by DO-248B DP14.

The first option put forth was in response to questions raised during the Preliminary Design Review, of whether it was possible for the nonshell code to overwrite any variable needed by the shell or otherwise interfere with the shell's operation. This proposal was to reduce the function of the shell to a minimal function of envelope checking and command limiting, and to add an on-access checksum or cyclic redundancy check scheme for the limited number of memory locations that would be required by the shell. This scheme would not prevent the nonshell code from performing any illegal access at the hardware level, but would detect any contamination of those memory locations and allow for a safe disengagement.

The second option put forth, and the one which is currently foremost in consideration, is to modify the 68040 Operation System (the piece of code in the 68040 that provides the low-level functionality for the board hardware, and calls the application software) to activate the 68040 Memory Management Unit, to provide a separate user space for the nonshell code. The MMU provides the ability to provide a limited memory map to the user space. Any illegal access would result in a memory fault which would be detected by the OS and handled appropriately.



Partitioning Options:

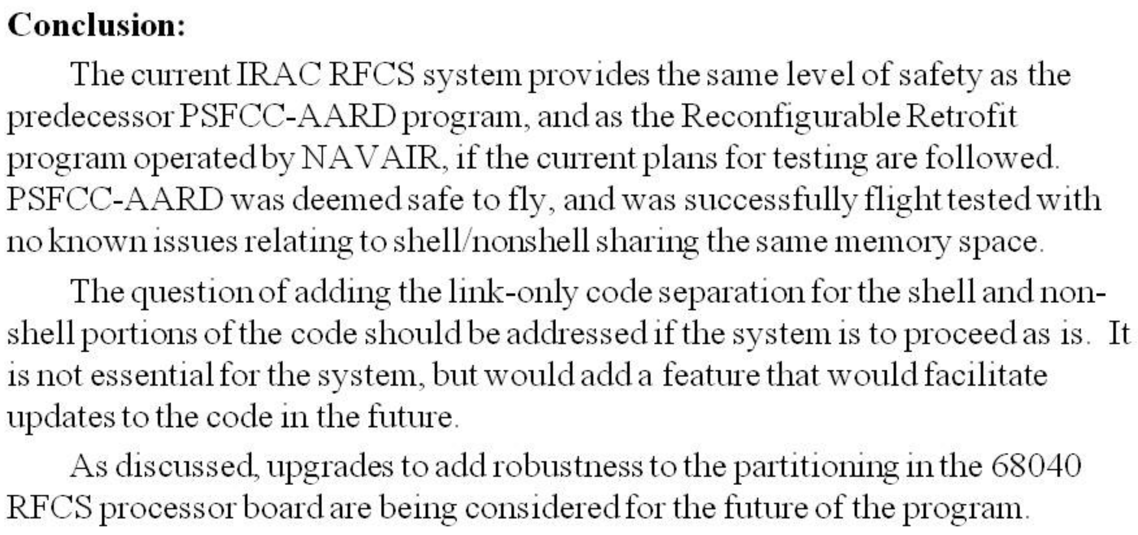
OPTIONS BEING CONSIDERED FOR CURRENT SYSTEM

Proposals are being investigated to add robust partitioning to the research processor:

“small shell” option – reduce function of shell to envelope checking and command limiting. Add checksum-protected RAM memory for any shell state variables. Disengage upon detection illegal access.

MMU option – Modify OS to activate MMU, providing a separate user space for the non-shell code. Illegal access will result in a memory fault detected by the 68040.

May 26, 2009 RFCS System Software and Testing Overview 10

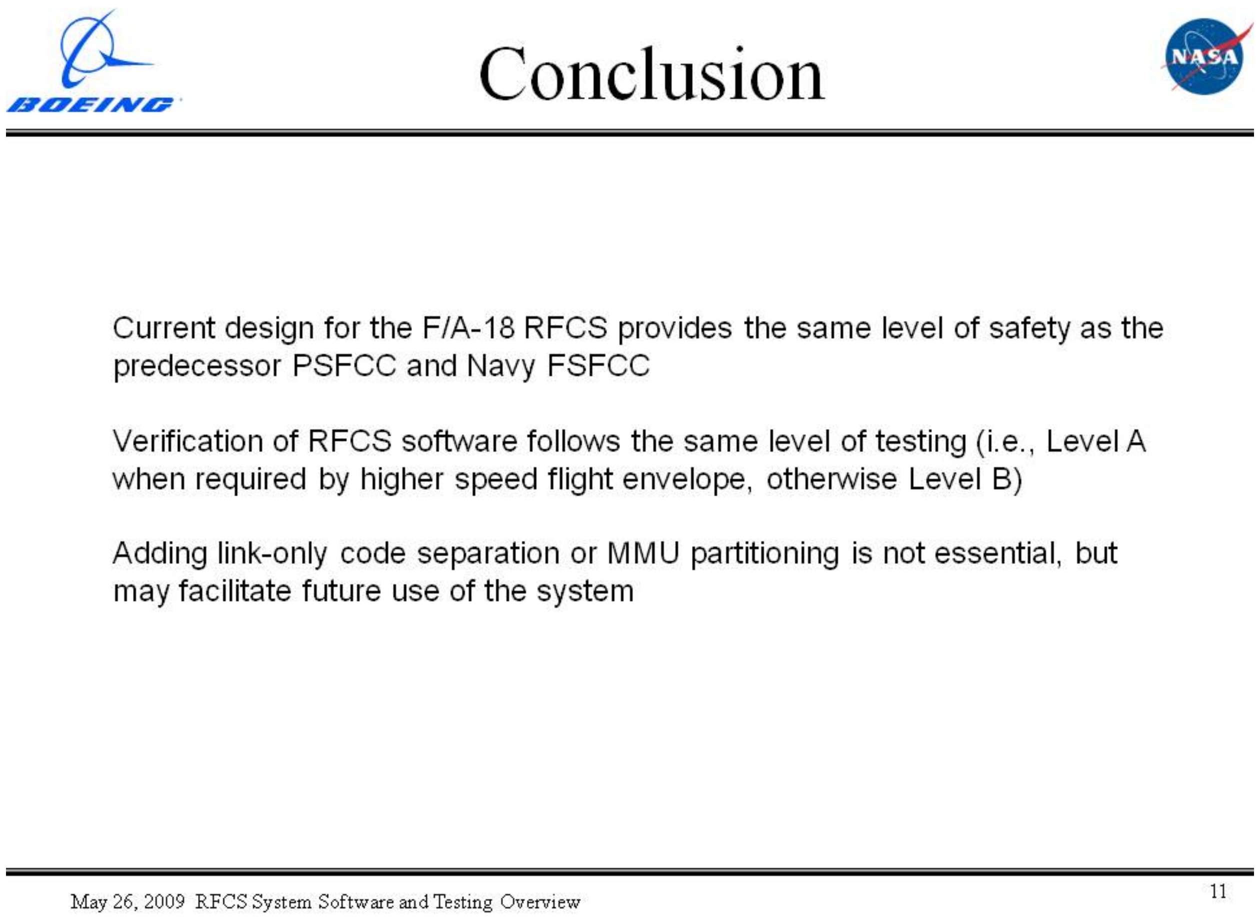


Conclusion:

The current IRAC RFCS system provides the same level of safety as the predecessor PSFCC-AARD program, and as the Reconfigurable Retrofit program operated by NAVAIR, if the current plans for testing are followed. PSFCC-AARD was deemed safe to fly, and was successfully flight tested with no known issues relating to shell/nonshell sharing the same memory space.

The question of adding the link-only code separation for the shell and non-shell portions of the code should be addressed if the system is to proceed as is. It is not essential for the system, but would add a feature that would facilitate updates to the code in the future.

As discussed, upgrades to add robustness to the partitioning in the 68040 RFCS processor board are being considered for the future of the program.



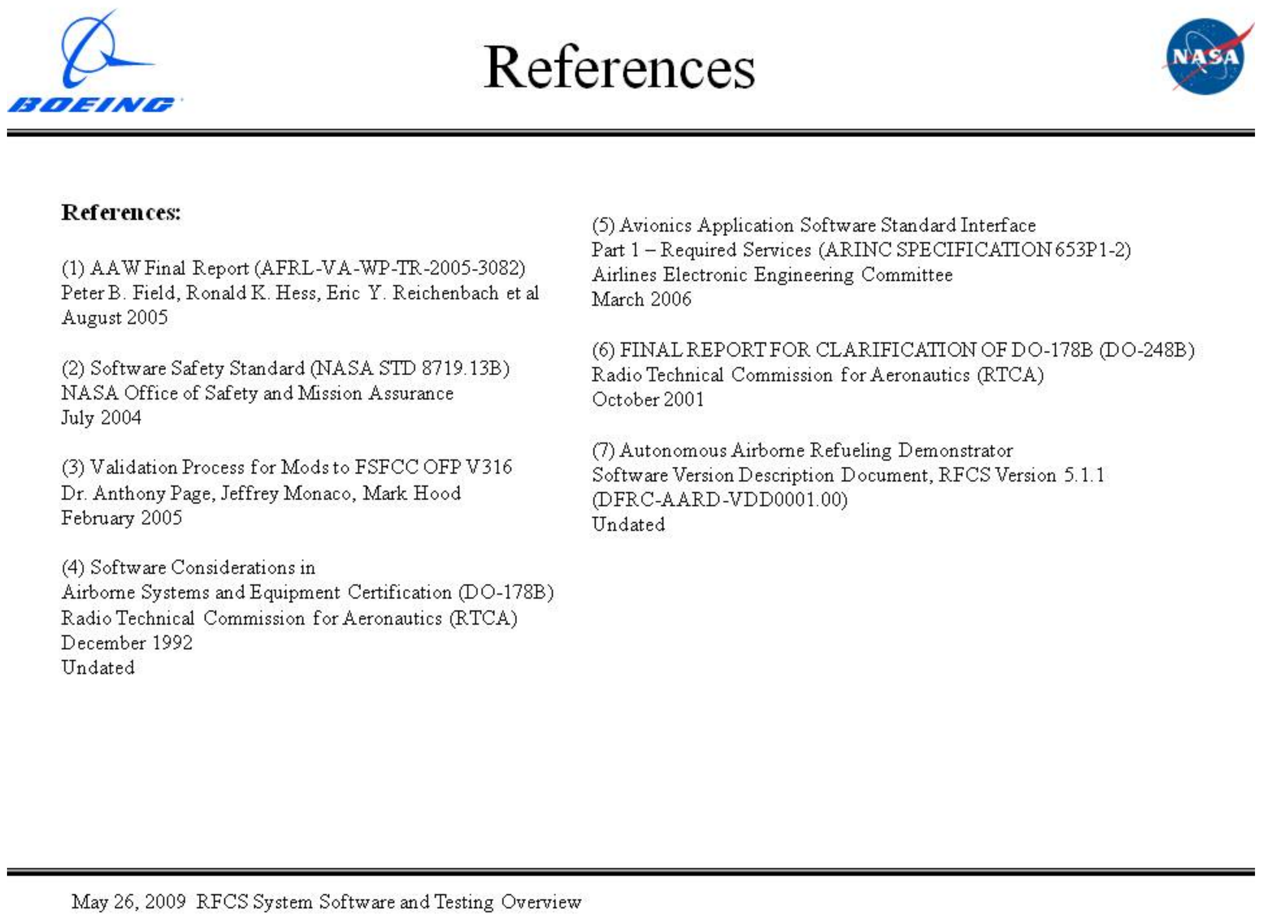
Conclusion

Current design for the F/A-18 RFCS provides the same level of safety as the predecessor PSFCC and Navy FSFCC

Verification of RFCS software follows the same level of testing (i.e., Level A when required by higher speed flight envelope, otherwise Level B)

Adding link-only code separation or MMU partitioning is not essential, but may facilitate future use of the system

May 26, 2009 RFCS System Software and Testing Overview 11



References

References:

(1) AAW Final Report (AFRL-VA-WP-TR-2005-3082)
Peter B. Field, Ronald K. Hess, Eric Y. Reichenbach et al
August 2005

(2) Software Safety Standard (NASA STD 8719.13B)
NASA Office of Safety and Mission Assurance
July 2004

(3) Validation Process for Mods to FSFCC OFP V316
Dr. Anthony Page, Jeffrey Monaco, Mark Hood
February 2005

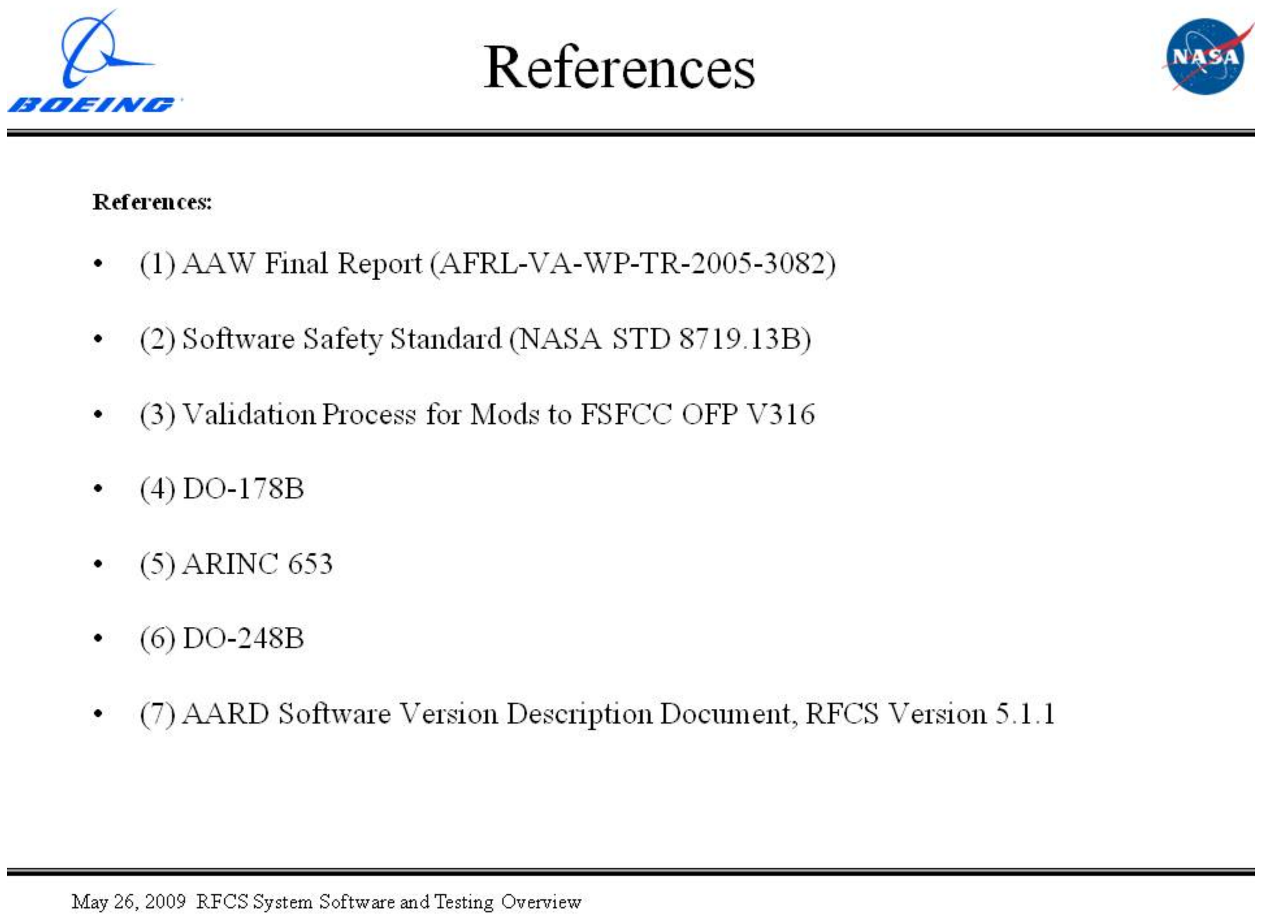
(4) Software Considerations in
Airborne Systems and Equipment Certification (DO-178B)
Radio Technical Commission for Aeronautics (RTCA)
December 1992
Undated

(5) Avionics Application Software Standard Interface
Part 1 – Required Services (ARINC SPECIFICATION 653P1-2)
Airlines Electronic Engineering Committee
March 2006

(6) FINAL REPORT FOR CLARIFICATION OF DO-178B (DO-248B)
Radio Technical Commission for Aeronautics (RTCA)
October 2001

(7) Autonomous Airborne Refueling Demonstrator
Software Version Description Document, RFCS Version 5.1.1
(DFRC-AARD-VDD0001.00)
Undated

May 26, 2009 RFCS System Software and Testing Overview



References

References:

- (1) AAW Final Report (AFRL-VA-WP-TR-2005-3082)
- (2) Software Safety Standard (NASA STD 8719.13B)
- (3) Validation Process for Mods to FSFCC OFP V316
- (4) DO-178B
- (5) ARINC 653
- (6) DO-248B
- (7) AARD Software Version Description Document, RFCS Version 5.1.1

May 26, 2009 RFCS System Software and Testing Overview